National Aeronautics and Space Administration

Basic Debugging

October 10, 2012

NASA Advanced Supercomputing Division

Outline

- Code Porting vs. Debugging
- Debugging with Intel Compiler Flags
  - Case study #1: -g -traceback -check -fpe0
  - Case study #2: non-deterministic behavior
  - Case study #3: incorrect algorithm, sensitivity to round-off
- Debugging with TotalView
  - Getting started with TotalView on Pleiades
  - Simple navigation with the GUI
  - Case study #4: program hang
  - Where to get more information on using TotalView

Question? Use the Webex chat facility to ask the Host

Code Porting vs. Debugging

Porting -- Getting a code, generally from somewhere else, to compile/build and generate "expected" results on the target machine. If the results are unexpected, then:

1. Compile with -fp-model precise
2. Lower compiler optimization: try compiling with -O1 or -O2 (default for Intel)
3. Try a different combination of compilers and libraries

There are 30+ compilers, 20+ MPI libraries, 6 NetCDF and 15 HDF libraries on Pleiades!

- Debugging in this webinar refers to debugging codes written in Fortran, C or C++
- Debugging may be necessary as part of the porting process

  Why? Because you are getting wrong or different or no results

  - The first two of these could be a machine precision issue, or a compiler optimization issue (See "Porting" above)
  - The 3rd could be due to a program hang, segfault, abort, etc.
- Debugging may be necessary if you are writing new code or modified an existing one

Question? Use the Webex chat facility to ask the Host

Debugging with Intel Compiler Flags

For Fortran use: -g -traceback -check -fpe0

For C/C++ use: -g -traceback

-g

- Tells compiler to generate full debugging information in the object file (.o file)
- Changes the default optimization to -O0, so need to explicitly add -O2 if no optimization level was previously specified
- Always compile with -g when using debuggers (i.e., TotalView)

-traceback

- Provides traceback information with source file, routine name, and line number when a severe error occurs at run time

```
forrtl: error (73): floating divide by zero
Image              PC                Routine            Line        Source
buggy              0000000000403144  sub1_                24  buggy.f
buggy              0000000000402ECF  MAIN__               13  buggy.f
buggy              0000000000402C0C  Unknown         Unknown  Unknown
libc.so.6          00007FFFECF2FBC6  Unknown         Unknown  Unknown
buggy              0000000000402B09  Unknown         Unknown  Unknown
```


Question? Use the Webex chat facility to ask the Host

Debugging with Intel Compiler Flags (2 of 3)

-check (same as -check all)

- checks for array bounds violation (same as -check bounds)
  - Example: dimension a(100) and the code uses a(101) = …
- checks for use of uninitialized variables (same as -check uninit)
  - Caution: the checking is very limited in scope
- checks for format, output_conversion, pointers, etc. … generally, unlikely to be the culprit
- Important: -check causes the program to run slow! Leave off after debugging

-fpe0

- traps floating-point exceptions, i.e., divide-by-zero, sqrt of negative, etc.
- when compiling with -fpe0, **all** source files need to be compiled with this flag (or with -fp-speculation=off) to avoid false-positives
  - Example:

```
if (z .ne. 0.0) then
   y = 1/z
else
   y = 1 + z
endif
```

both branches are executed simultaneously and one branch is discarded after evaluating the if conditional

Question? Use the Webex chat facility to ask the Host

Debugging with Intel Compiler Flags (3 of 3)

Example of array bounds violation message

```
forrtl: severe (408): fort: (2): Subscript #1 of the array Y has value 101 which
is greater than the upper bound of 100
```

Image	PC	Routine	Line	Source
buggy	0000000000046AFCA	Unknown	Unknown	Unknown
buggy	0000000000469AC6	Unknown	Unknown	Unknown
buggy	0000000000421DE0	Unknown	Unknown	Unknown
buggy	0000000000404B6E	Unknown	Unknown	Unknown
buggy	0000000000405091	Unknown	Unknown	Unknown
buggy	00000000004033D9	sub1_	27	buggy.f
buggy	0000000000402F82	MAIN__	13	buggy.f
buggy	0000000000402C0C	Unknown	Unknown	Unknown
libc.so.6	00007FFFECF2FBC6	Unknown	Unknown	Unknown
buggy	0000000000402B09	Unknown	Unknown	Unknown

Question? Use the Webex chat facility to ask the Host

Case Study #1: Program buggy

```
      program buggy
      parameter (nmax = 100)
      real (kind=8) :: a(nmax), b(nmax), c

      a = 2.0   ! sets all elements of a() to 2.0
      b = 3.0   ! sets all elements of b() to 3.0

      print *, 'a(1) + a(2) = ', a(1) + a(2)
      print *, 'b(1) + b(2) = ', b(1) + b(2)

      b(3) = a(3) + c

      call sub1(nmax,a,b,c)

      print *, 'after calling sub1'
      print *, 'a(1) + a(2) = ', a(1) + a(2)
      print *, 'b(1) + b(2) = ', b(1) + b(2)
      end

      subroutine sub1(n,x,y,z)
      real (kind = 8) :: x(n), y(n), z

      y(3) = x(3) + z  ! note uninit doesn't trap here
      y(4) = 1.0/z

      do i = 10,200
        y(i) = dfloat(i)
        x(i) = 1.0/dfloat(i)
      enddo

      end
```


Question? Use the Webex chat facility to ask the Host

Case Study #1: Initial run and debug

```
pfe8> ifort -o buggy buggy.f
pfe8> buggy
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
 after calling sub1
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =   1.9704911667637 35E-002
< ...program hangs >
```

Note wrong answer after calling sub1 and program hang.

```
pfe8> ifort -o buggy -g -traceback -check -fpe0 buggy.f
pfe8> buggy
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
forrtl: severe (193): Run-Time Check Failure. The variable 'buggy_$C' is being u
sed without being defined
Image              PC                Routine            Line        Source

buggy              000000000046B20A  Unknown               Unknown  Unknown
buggy              0000000000469D06  Unknown               Unknown  Unknown
buggy              0000000000422020  Unknown               Unknown  Unknown
buggy              0000000000404DAE  Unknown               Unknown  Unknown
buggy              0000000000405C88  Unknown               Unknown  Unknown
buggy              0000000000402F6F  MAIN__                     11  buggy.f
buggy              0000000000402C0C  Unknown               Unknown  Unknown
libc.so.6          00007FFFECF2FBC6  Unknown               Unknown  Unknown
buggy              0000000000402B09  Unknown               Unknown  Unknown
pfe8>
```

forrtl is the Fortran Runtime Library (RTL)

Note variable name and offending line number in the traceback.

Question? Use the Webex chat facility to ask the Host

Case Study #1: Modify code and re-run

```
 1       program buggy
 2       parameter (nmax = 100)
 3       real (kind=8) :: a(nmax), b(nmax), c
 4
 5       a = 2.0   ! sets all elements of a() to 2.0
 6       b = 3.0   ! sets all elements of b() to 3.0
 7
 8       print *, 'a(1) + a(2) = ', a(1) + a(2)
 9       print *, 'b(1) + b(2) = ', b(1) + b(2)
10
11 !     b(3) = a(3) + c
12
13       call sub1(nmax,a,b,c)
14
```

Need to assign a value to "c" before use, but for now just comment line 11 out and continue.

```
pfe8> vi buggy.f
pfe8> !ifort
ifort -o buggy -g -traceback -check -fpe0 buggy.f
pfe8> buggy
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
forrtl: error (73): floating divide by zero
Image              PC                Routine            Line        Source

buggy              0000000000403423  sub1_                24  buggy.f
buggy              0000000000402F82  MAIN__               13  buggy.f
buggy              0000000000402C0C  Unknown         Unknown  Unknown
libc.so.6          00007FFFECF2FBC6  Unknown         Unknown  Unknown
buggy              0000000000402B09  Unknown         Unknown  Unknown
Abort (core dumped)
pfe8>
```

Recompile and re-run. The code now aborts with a divide-by-zero (caught by -fpe0) at line 24.

Question? Use the Webex chat facility to ask the Host

Case Study #1: Modify code and re-run

```
 1        program buggy
 2        parameter (nmax = 100)
 3        real (kind=8) :: a(nmax), b(nmax), c
 4
 5        a = 2.0   ! sets all elements of a() to 2.0
 6        b = 3.0   ! sets all elements of b() to 3.0
 7
 8        print *, 'a(1) + a(2) = ', a(1) + a(2)
 9        print *, 'b(1) + b(2) = ', b(1) + b(2)
10
11 !      b(3) = a(3) + c
12
13        call sub1(nmax,a,b,c)
14
15        print *, 'after calling sub1'
16        print *, 'a(1) + a(2) = ', a(1) + a(2)
17        print *, 'b(1) + b(2) = ', b(1) + b(2)
18        end
19
20        subroutine sub1(n,x,y,z)
21        real (kind = 8) :: x(n), y(n), z
22
23        y(3) = x(3) + z  ! note uninit doesn't trap here
24        y(4) = 1.0/z
25
26        do i = 10,200
27          y(i) = dfloat(i)
28          x(i) = 1.0/dfloat(i)
29        enddo
30
31        end
```

"c" wasn't assigned a value, so it turned out to be 0.0 (but you can't always depend on uninitialized variables having a value of zero!).

"c" was passed into sub1, so the dummy argument "z" has a value of 0 -- triggering the divide-by-zero at line 24. But WAIT ... note that the RTL didn't catch that "z" was being used at line 23 even though it hasn't been initialized!

Question? Use the Webex chat facility to ask the Host

Case Study #1: Modify code and re-run

```
 20       subroutine sub1(n,x,y,z)
 21       real (kind = 8) :: x(n), y(n), z
 22
 23       y(3) = x(3) + z  ! note uninit doesn't trap here
 24 !     y(4) = 1.0/z
 25
 26       do i = 10,200
 27         y(i) = dfloat(i)
 28         x(i) = 1.0/dfloat(i)
 29       enddo
 30
 31       end
```

```
pfe8> !ifort
ifort -o buggy -g -traceback -check -fpe0 buggy.f
pfe8> buggy
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
forrtl: severe (408): fort: (2): Subscript #1 of the array Y has value 101 which
 is greater than the upper bound of 100

Image              PC                Routine            Line        Source

buggy              000000000046B10A  Unknown               Unknown  Unknown
buggy              0000000000469C06  Unknown               Unknown  Unknown
buggy              0000000000421F20  Unknown               Unknown  Unknown
buggy              0000000000404CAE  Unknown               Unknown  Unknown
buggy              00000000004051D1  Unknown               Unknown  Unknown
buggy              0000000000403515  sub1_                    27    buggy.f
buggy              0000000000402F82  MAIN__                   13    buggy.f
buggy              0000000000402C0C  Unknown               Unknown  Unknown
libc.so.6          00007FFFECF2FBC6  Unknown               Unknown  Unknown
buggy              0000000000402B09  Unknown               Unknown  Unknown
pfe8>
```

Need to fix the divide-by-zero, but for now just comment it out and continue.

Arrays a and b, which are passed into sub1 as x and y, were declared with dimension NMAX=100.

If y were a multi-dimensional array, the forrtl complaint could point to subscript #2 or #3, for example.

Question? Use the Webex chat facility to ask the Host

Case Study #1: Modify code and re-run

```
 20        subroutine sub1(n,x,y,z)
 21        real (kind = 8) :: x(1), y(1), z
 22
 23        y(3) = x(3) + z  ! note uninit doesn't trap here
 24 !      y(4) = 1.0/z
 25
 26        do i = 10,200
 27          y(i) = dfloat(i)
 28          x(i) = 1.0/dfloat(i)
 29        enddo
 30
 31        end
```

```
pfe8> !ifort
ifort -o buggy -g -traceback -check -fpe0 buggy.f
pfe8> buggy
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
pfe8>
```

```
pfe8> !ifort
ifort -o buggy -g -traceback -check -fpe0 buggy.f
pfe8> buggy
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
 after calling sub1
 a(1) + a(2) =    4.00000000000000
 b(1) + b(2) =    6.00000000000000
pfe8>
```

Need to change the limits of the DO loop at line 26 or increase NMAX, but let's see what happens when the arrays x and y are dimensioned 1 (or *) as in many legacy codes. This is perfectly legal Fortran, btw.

The -check flag no longer catches array bounds violation on arrays x and y and the program ends without errors, but the result is unexpected.

This is the correct behavior after the remaining array bounds violation bugs have been removed.

Question? Use the Webex chat facility to ask the Host

Case Study #1: Take home

- If arrays are dimension 1 or *, for example, a(1), x(1,nj), then array bounds checking on those arrays is ineffective.
- Checking of uninitialized variables is limited in scope.
  (However, check back for the next webinar on "Uninit," which provides a technique for trapping all uninitialized variables)
- Debugging with compiler debug options is useful for catching most common bugs, but **not** all bugs. So, just because a code has passed a "health check" using the compiler debug options does not mean it's bug-free.
- Debugging with compiler flags is an iterative process:
  compile → run → modify → compile → run → ...
- To streamline the iterative process, get an interactive PBS session:
  qsub -I -q devel -lselect=NN:ncpus=YY,walltime=2:00:00
  and make successive runs in the same PBS session.
  To avoid typing all the commands in runscript, just make it executable (chmod u+x runscript) and run interactively:
  ./runscript

Question? Use the Webex chat facility to ask the Host

Case Study #2: “Magic Trick”

```
      program main
! compile with pgf90
      integer m,n
      m = 5
      n = 7

      call switcha(m, n, .True.)
      call printa(m, n, .True.)

      end

      subroutine switcha(m, n, iflag)
      logical iflag
      integer m,n,k

      if (iflag) then
        k=m
        m=n
        n=k
        iflag = .False.
      endif

      end

      subroutine printa(m, n, iflag)
      logical iflag
      if (iflag) then
        print *, 'm = ',m
      else
        print *, 'n = ',n
      endif
      end
```


Question? Use the Webex chat facility to ask the Host

Case Study #2: Non-deterministic behavior

```
cfe2> ifort -o ex1 tv_ex1.f
cfe2> ex1
 n =           5
cfe2> module list
Currently Loaded Modulefiles:
  1) intel-comp.11.1.072
cfe2> module switch intel-comp.11.1.072 intel-comp.10.1.013
cfe2> !ifort
ifort -o ex1 tv_ex1.f
cfe2> ex1
 m =           7
```

Different versions of ifort give different answers on Columbia.

Code aborts with a segfault with Intel compiler and gives one of the two Columbia results with PGI compiler.

```
pfe8> ifort -o ex1 tv_ex1.f
pfe8> ex1
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC                Routine            Line        Source

ex1                0000000000402C8E  Unknown               Unknown  Unknown
ex1                0000000000402C0C  Unknown               Unknown  Unknown
libc.so.6          00007FFFED133BC6  Unknown               Unknown  Unknown
ex1                0000000000402B09  Unknown               Unknown  Unknown
pfe8> module purge
pfe8> module load comp-pgi/12.5
pfe8> pgf90 -o ex1_pg tv_ex1.f
pfe8> ex1_pg
 n =           5
```

Take home:

- Bugs can cause different results with different compilers, different machines, different compiler options, etc.
- Just because you get the same results doesn't mean the code is bug-free!

Question? Use the Webex chat facility to ask the Host

Case Study #2: Bug Revealed

```
pfe8> ifort -o ex1_dbg -g -traceback -check -fpe0 tv_ex1.f
pfe8> ex1_dbg
forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC                Routine            Line        Source

ex1_dbg            0000000000402D24  switcha_                 20  tv_ex1.f
ex1_dbg            0000000000402C92  MAIN__                    6  tv_ex1.f
ex1_dbg            0000000000402C0C  Unknown                 Unknown  Unknown
libc.so.6          00007FFFECF2FBC6  Unknown                 Unknown  Unknown
ex1_dbg            0000000000402B09  Unknown                 Unknown  Unknown
```

```
 1        program main
 2        integer m,n
 3        m = 5
 4        n = 7
 5
 6        call switcha(m, n, .True.)
 7
 8        call printa(m, n, .True.)
 9
10        end
11
12        subroutine switcha(m, n, iflag)
13        logical iflag
14        integer m,n,k
15
16        if (iflag) then
17          k=m
18          m=n
19          n=k
20          iflag = .False.
21        endif
```

The segfault with ifort on Pleiades is a clue that there is a bug in the code, so compile with debug flags.

Fortran passes by address into routines, so iflag in switcha contains the memory address of the literal constant .True., and, thus, it cannot be overwritten at line 20.

Question? Use the Webex chat facility to ask the Host

Case Study #2: Simpler “Magic Trick”

```
pfe8> cat tv_ex2.f
      program main
      integer i,j

      call first(1,j)
      call second(1)
      end

      subroutine first(i,j)
      integer i,j
      i = 5
      j = 7
      end

      subroutine second(k)
      integer k
      print *, 'one = ',k
      end
pfe8> pgf90 -o ex2_pg tv_ex2.f
pfe8> ex2_pg
 one =             5
```


Question? Use the Webex chat facility to ask the Host

Case Study #3: Improper algorithm or sensitivity to round-off error

Write a program to evaluate:

$$\lim_{x \to \infty} \left\{ \sqrt{x^2 + 3x} - \sqrt{x^2 + 2x} \right\}$$

```
      program main
! estimate limit as x approaches infinity
      real (kind=8) :: x, y

      x = 1.0d0
      do i = 1,30
         y = sqrt(x*x + 3.248d0*x) - sqrt(x*x + 2.d0*x)
         print *, 'x,y = ',x,y
         x = x*10.0d0
      enddo

      end
```

Note: 3 has been changed to 3.248 to show loss of significant figures in the results.

Question? Use the Webex chat facility to ask the Host

Case Study #3: Results

```
pfe8> ifort -o xinf xinf.f
pfe8> xinf
 x,y =    1.00000000000000        0.329016876909243
 x,y =    10.0000000000000        0.555544505847205
 x,y =    100.000000000000        0.615973640458648
 x,y =    1000.00000000000        0.623182949835950
 x,y =    10000.0000000000        0.623918147612130
 x,y =    100000.000000000        0.623991813277826
 x,y =    1000000.00000000        0.623999181319959
 x,y =    10000000.0000000        0.623999917879701
 x,y =    100000000.000000        0.623999983072281
 x,y =    1000000000.00000        0.623999953269958
 x,y =    10000000000.0000        0.623998641967773
 x,y =    100000000000.000        0.624008178710938
 x,y =    1000000000000.00        0.624023437500000
 x,y =    10000000000000.0        0.623046875000000
 x,y =    100000000000000.        0.640625000000000
 x,y =   1.00000000000000E+015   0.625000000000000
 x,y =   1.00000000000000E+016   0.000000000000000E+000
 x,y =   1.00000000000000E+017   0.000000000000000E+000
 x,y =   1.00000000000000E+018   0.000000000000000E+000
 x,y =   1.00000000000000E+019   0.000000000000000E+000
 x,y =   1.00000000000000E+020   0.000000000000000E+000
 x,y =   1.00000000000000E+021   0.000000000000000E+000
 x,y =   1.00000000000000E+022   0.000000000000000E+000
 x,y =   9.99999999999999E+022   0.000000000000000E+000
 x,y =   1.00000000000000E+024   0.000000000000000E+000
 x,y =   9.99999999999999E+024   0.000000000000000E+000
 x,y =   9.99999999999999E+025   0.000000000000000E+000
 x,y =   9.99999999999999E+026   0.000000000000000E+000
 x,y =   1.00000000000000E+028   0.000000000000000E+000
 x,y =   9.99999999999999E+028   0.000000000000000E+000
```


Question? Use the Webex chat facility to ask the Host

Case Study #3: Take home

Analytical answer: $$\left\{\sqrt{x^2+3x}-\sqrt{x^2+2x}\right\}\frac{\left(\sqrt{x^2+3x}+\sqrt{x^2+2x}\right)}{\left(\sqrt{x^2+3x}+\sqrt{x^2+2x}\right)}=\frac{x}{\left(\sqrt{x^2+3x}+\sqrt{x^2+2x}\right)}$$

$$\lim_{x\to\infty}\left\{\sqrt{x^2+3x}-\sqrt{x^2+2x}\right\}=1/2 \qquad \lim_{x\to\infty}\left\{\sqrt{x^2+3.248x}-\sqrt{x^2+2x}\right\}=0.624$$

- Note that for $x > 10^{16}$ the 3.248x and 2x terms under the square root get dropped off compared to x^2.
- The correct answer, 0.624 to 15 significant figures, is only obtained for a very narrow range in the value of x (if at all) for this program.
- If the formula used in the program is changed to the one above on the right hand side, then the correct answer can be obtained for any $x > 10^{16}$.
- Is the issue here one of round-off errors or incorrect formula?
- Part of debugging is figuring out when the answers are as good as they are going to get due to sensitivities to round-off error, unstable algorithm, etc.

Question? Use the Webex chat facility to ask the Host

Debugging with TotalView

Question? Use the Webex chat facility to ask the Host

Getting started with TotalView on Pleiades

1. Make sure that you can display X11 graphics from pfe
   - log in with ssh -X or add “ForwardX11 yes” to your .ssh/config file on your workstation
   - echo $DISPLAY should show some setting for the DISPLAY environment set by ssh
   - test by running xclock
   - may need to use VNC if interactive response is very slow
2. Compile your code with the -g compiler flag
3. Submit an interactive PBS job and forward your DISPLAY environment
   - qsub -I -v DISPLAY -q devel -lselect=N:ncpus=XX,walltime=HH:MM:SS
4. Once the PBS job has started, load the totalview module if it is not automatically loaded from your .login or .cshrc file
   - module load totalview/8.9.2-1
5. Start totalview with either:

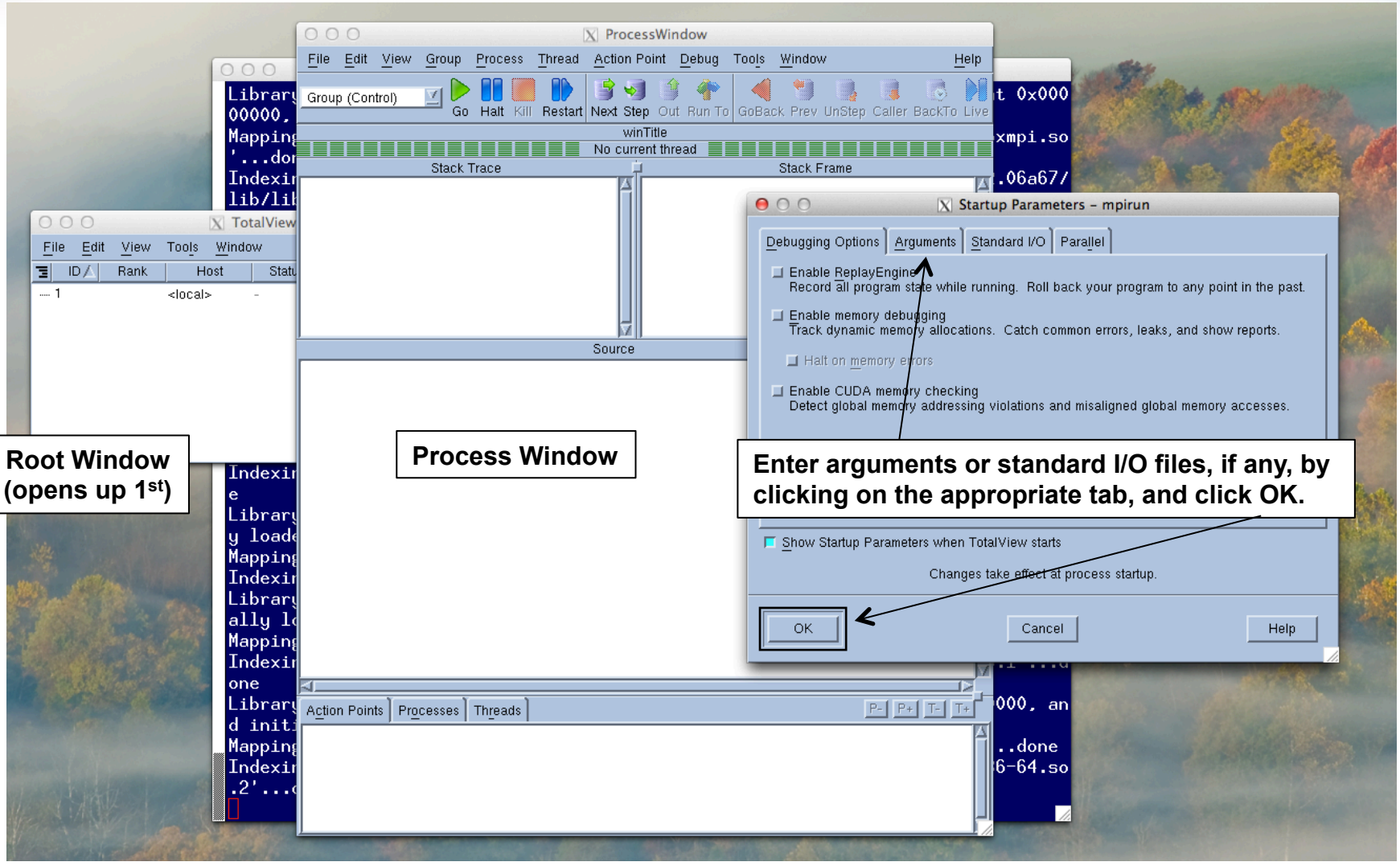
   totalview ./exe [-a args] for serial or OpenMP programs

   or

   mpiexec_mpt -tv -np <nprocs> ./exe for MPI or hybrid programs (assumes MPT)

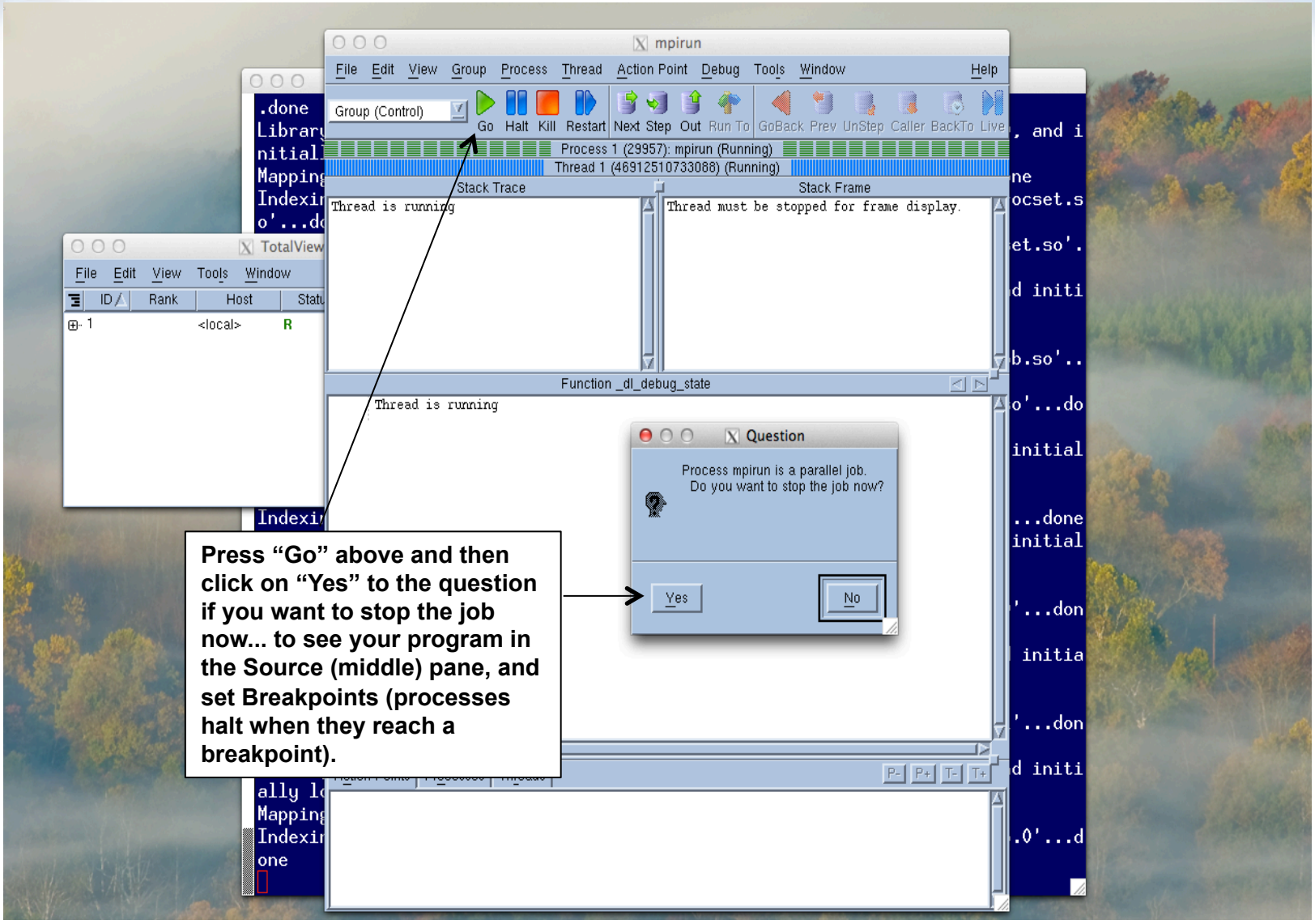
Question? Use the Webex chat facility to ask the Host

TotalView Start-up View



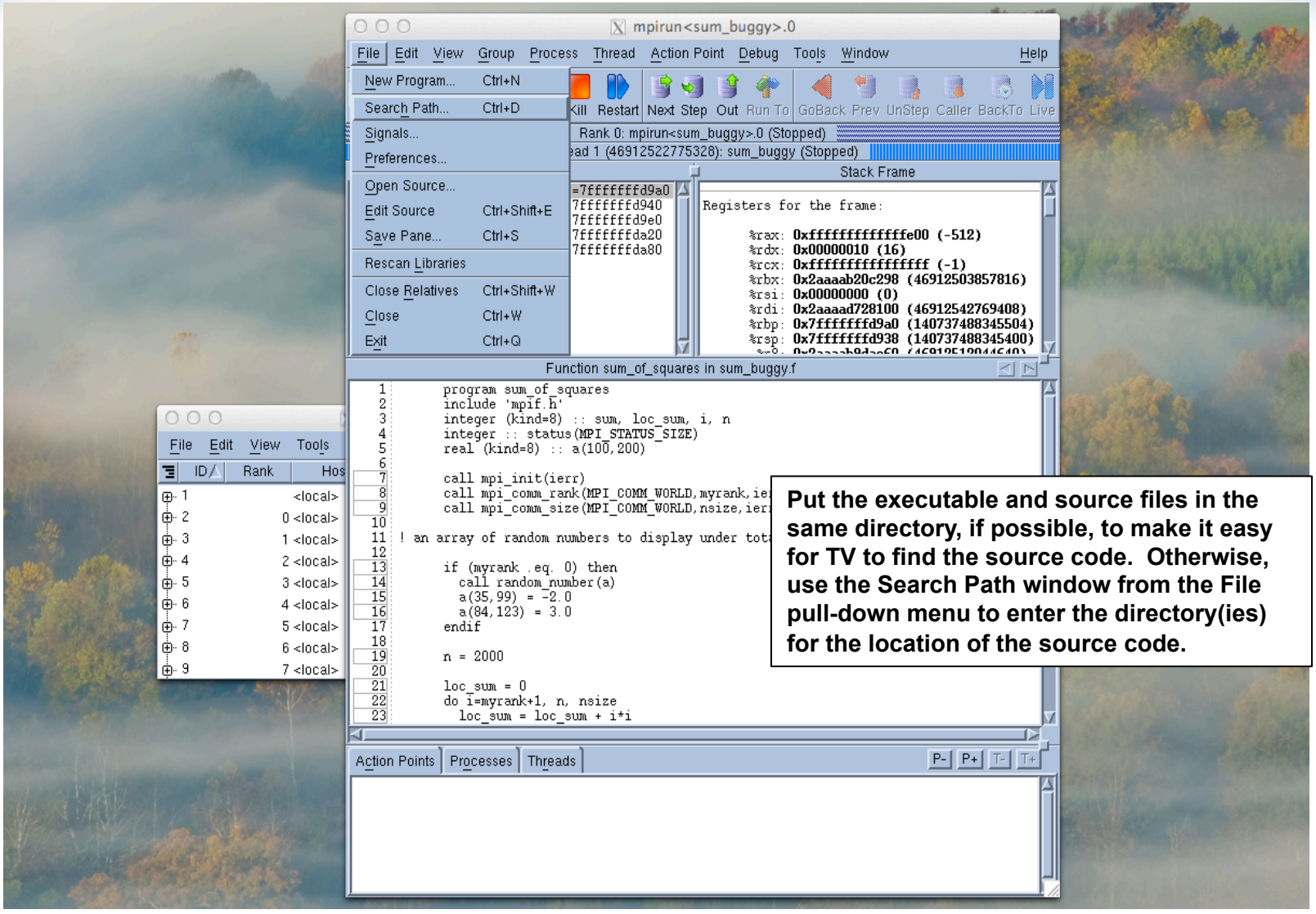
Question? Use the Webex chat facility to ask the Host

Simple Navigation with the GUI



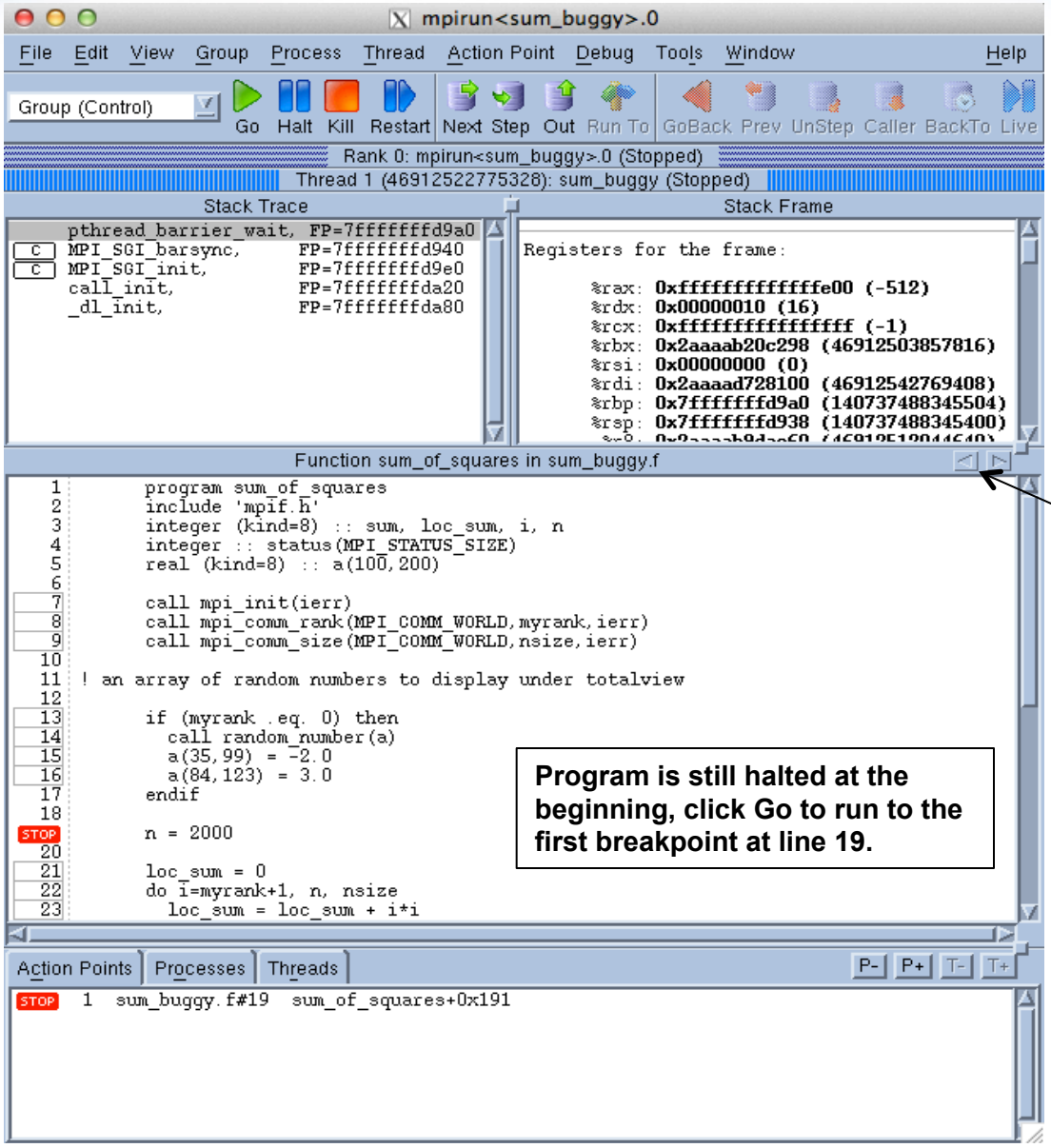
Question? Use the Webex chat facility to ask the Host

Search Path for Source Code



Question? Use the Webex chat facility to ask the Host

Navigate in Process Window



Go -- starts the program running
Halt -- stops it
Kill -- kills all the processes
Restart -- restarts from the beginning

Next -- goes to the next line in the program even if it is a function or routine
Step -- same as Next except that it will step into the function or routine
Out -- runs to the completion of the function or routine and steps out to the caller.

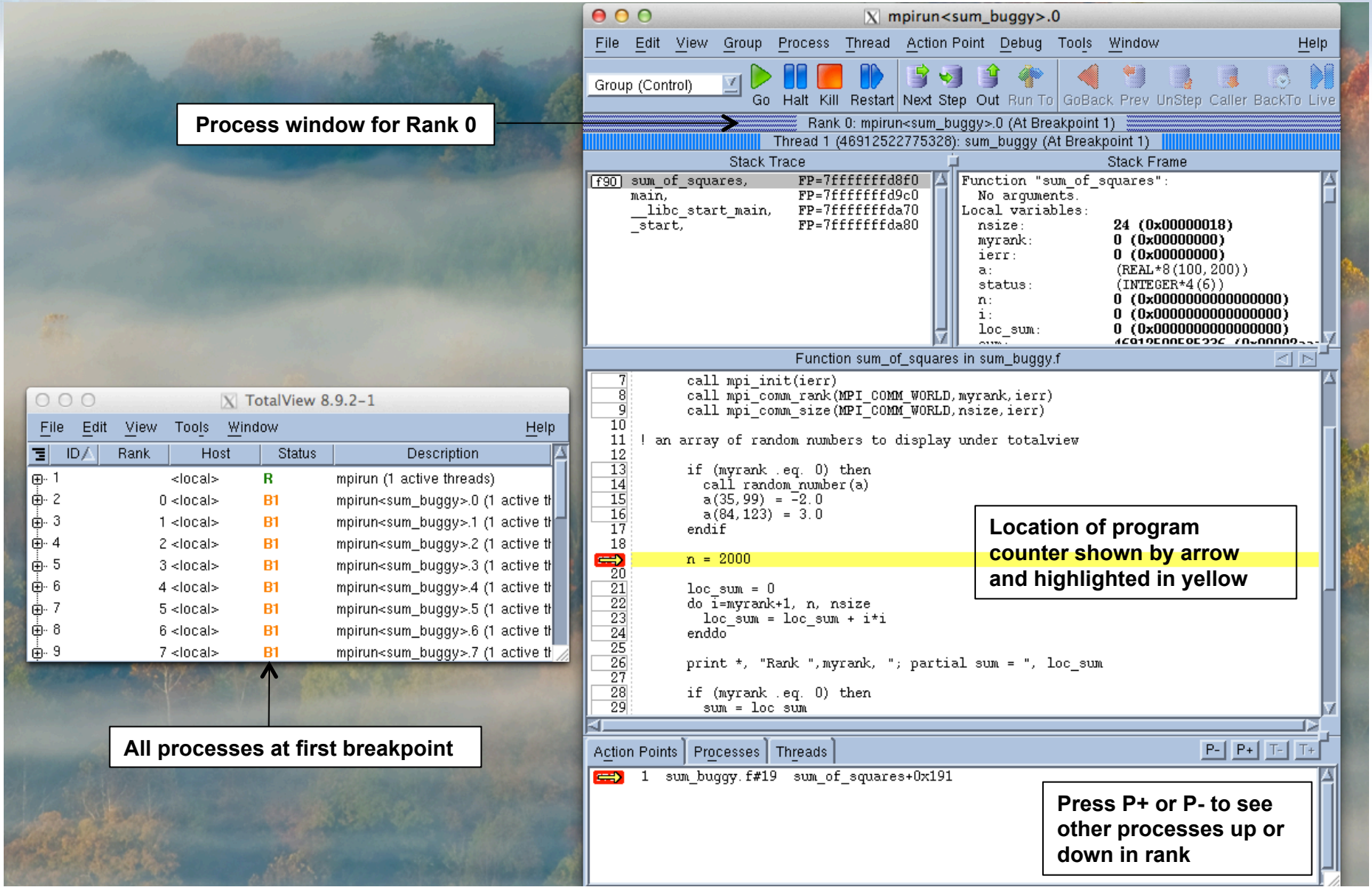
Arrows navigate the view of the source code up or down the call graph.

Breakpoints can be set at any location containing a rectangular box by clicking on the box, or via the Action Point pull-down menu above.

Program is still halted at the beginning, click Go to run to the first breakpoint at line 19.

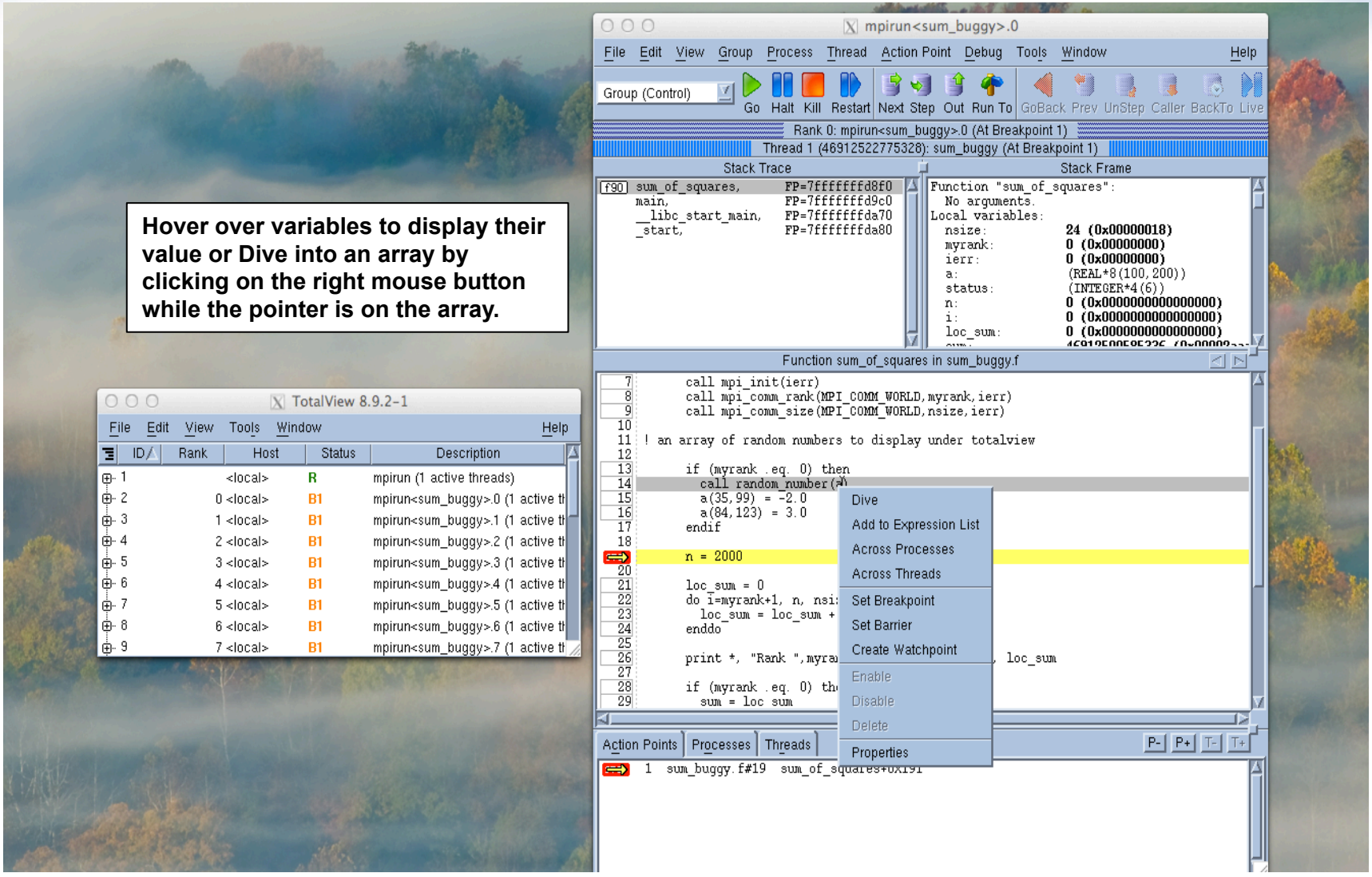
Question? Use the Webex chat facility to ask the Host

At first breakpoint



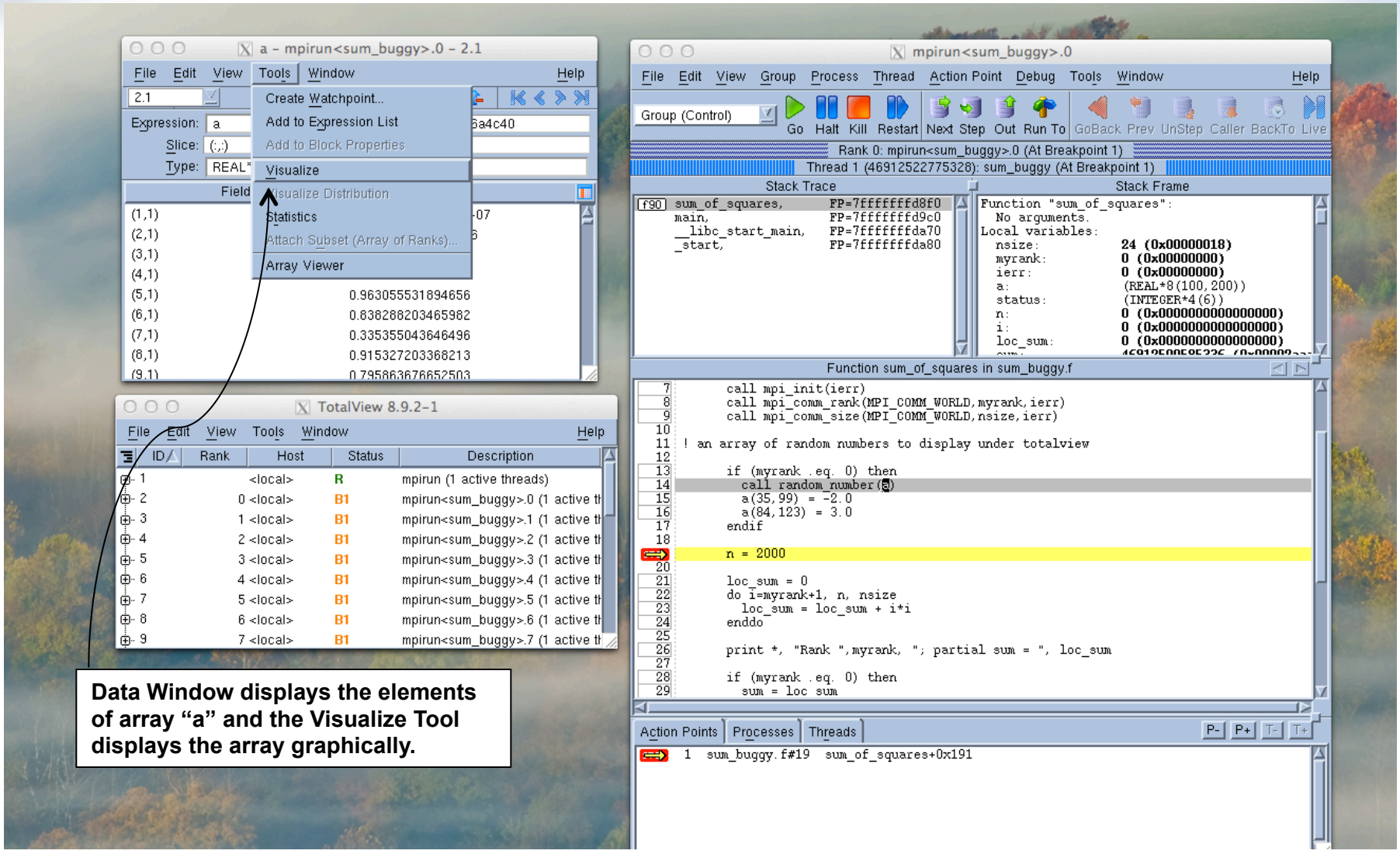
Question? Use the Webex chat facility to ask the Host

Diving



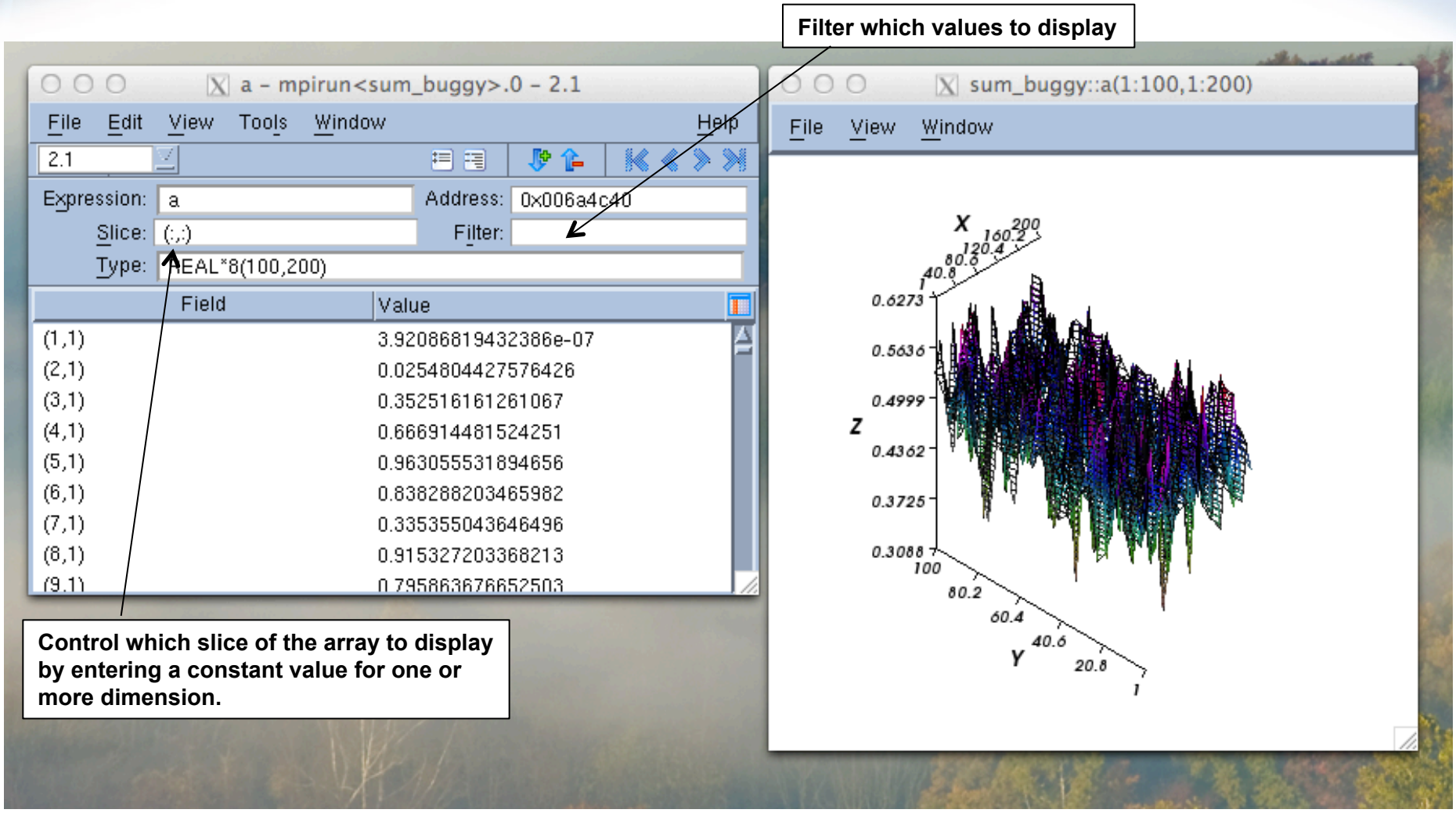
Question? Use the Webex chat facility to ask the Host

Data Display



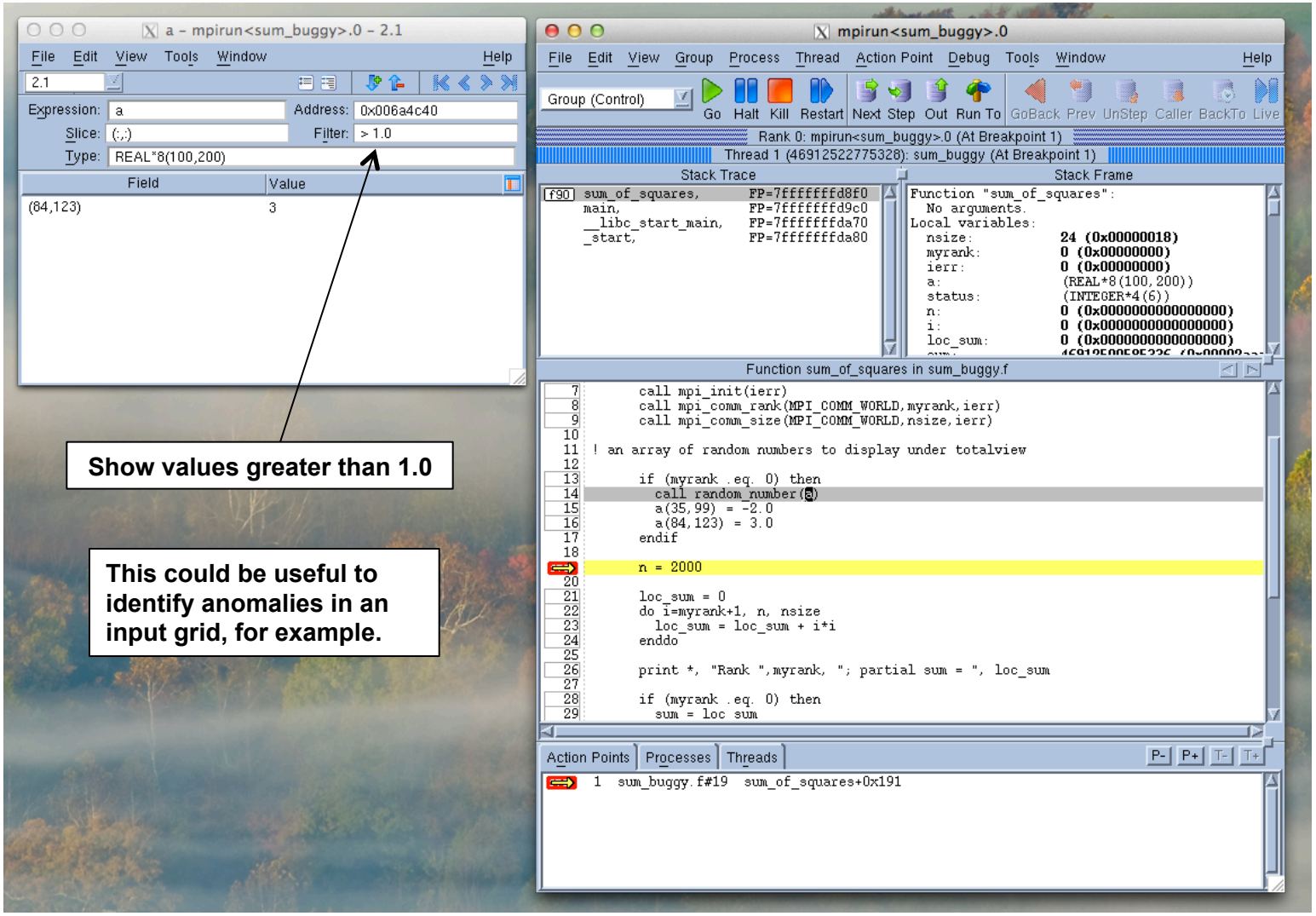
Question? Use the Webex chat facility to ask the Host

Visualizer, Data Slicer, and Filter



Question? Use the Webex chat facility to ask the Host

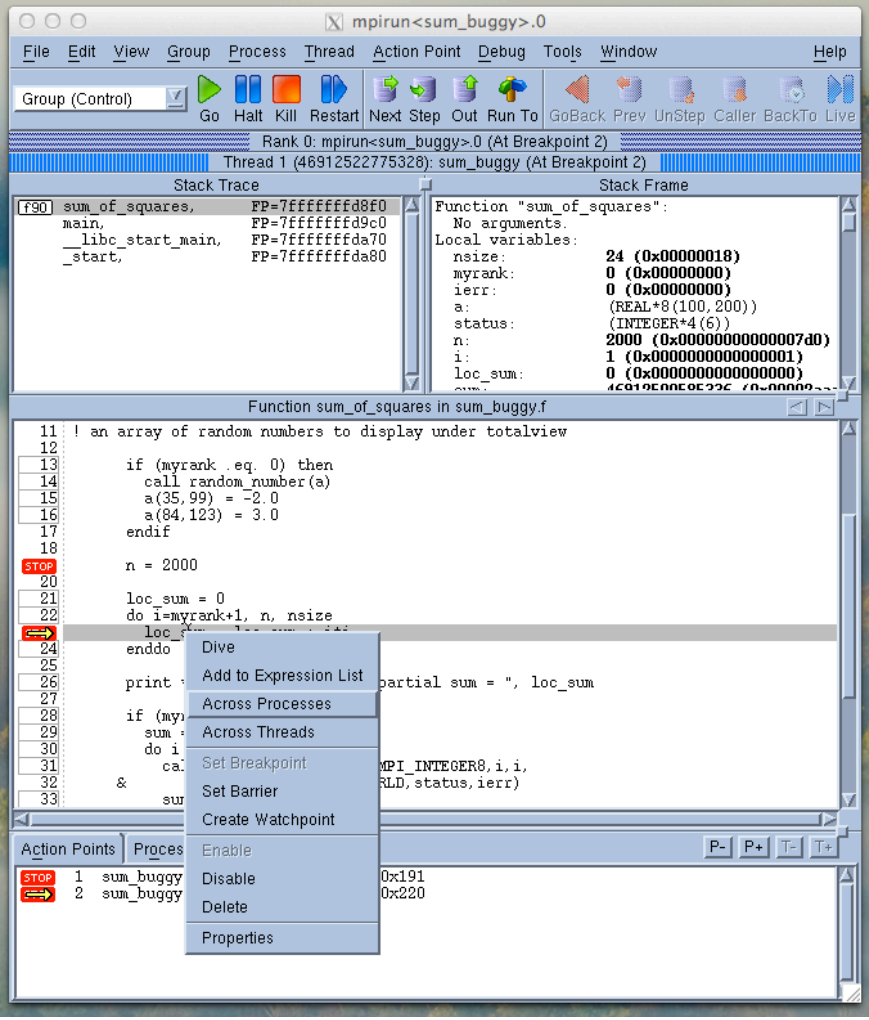
Result of Data Filter



Question? Use the Webex chat facility to ask the Host

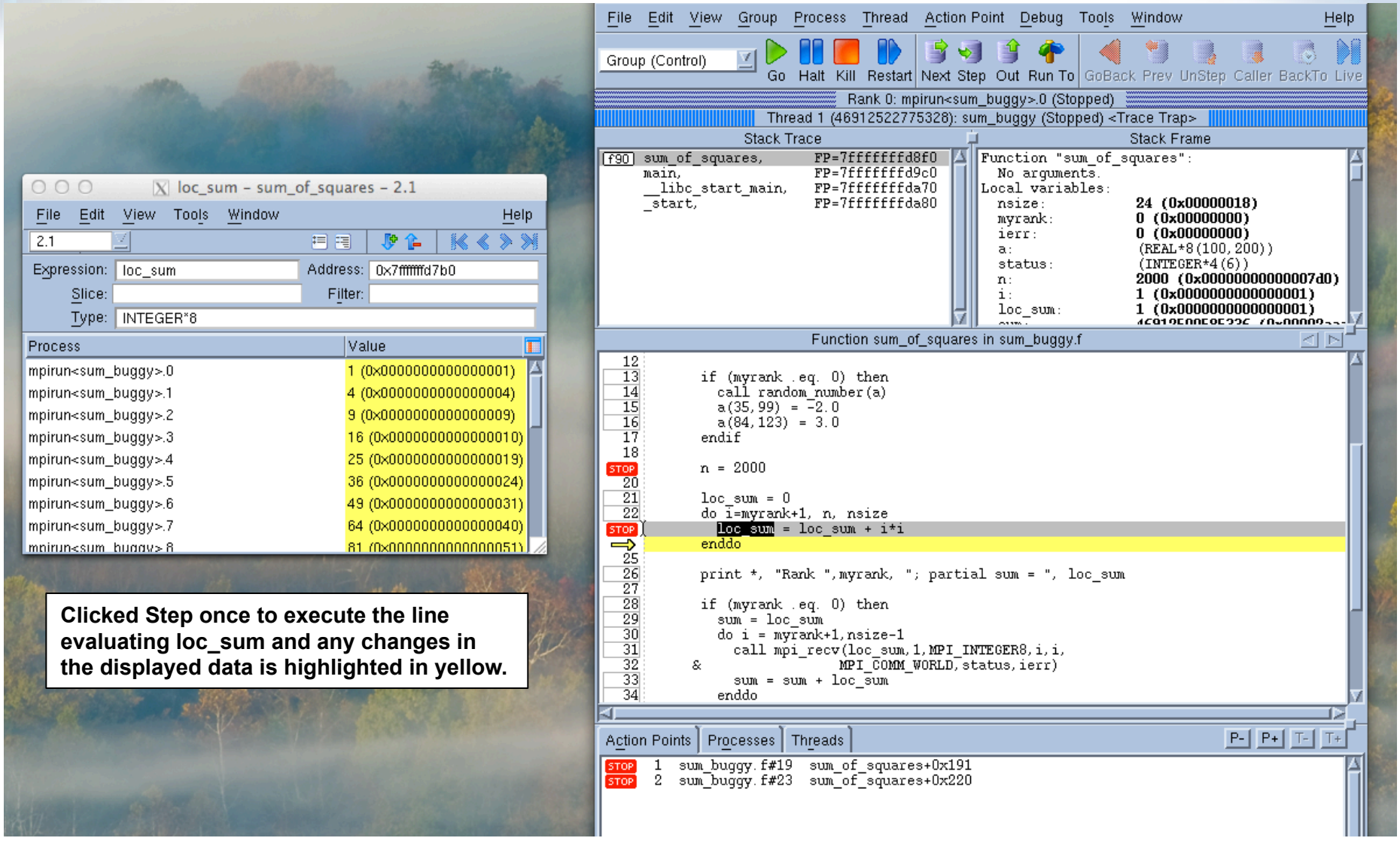
Viewing data across processes

Set a second breakpoint, press “Run To” second breakpoint, and display value of loc_sum across all processes.



Question? Use the Webex chat facility to ask the Host

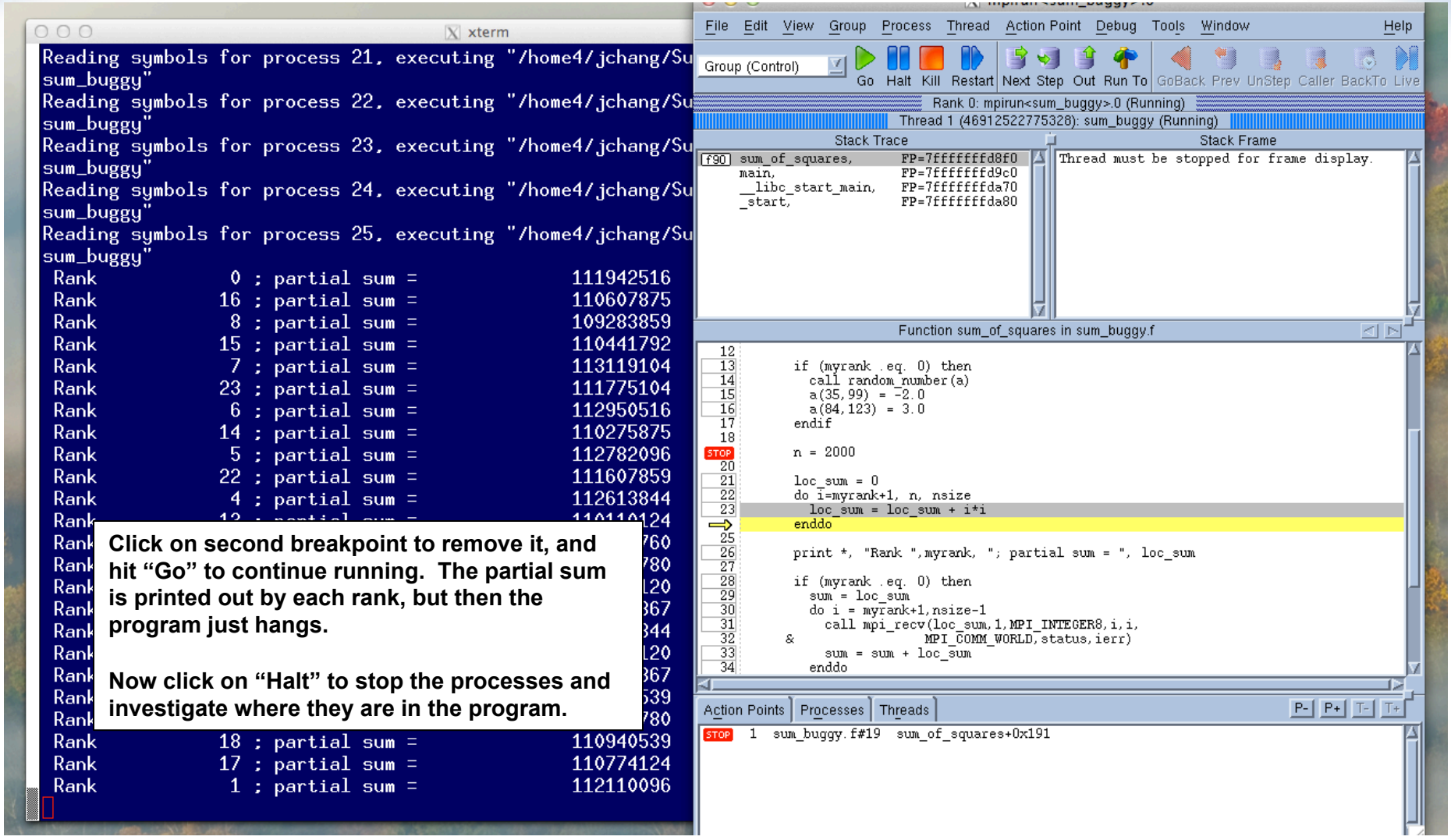
Values that change are highlighted



Clicked Step once to execute the line evaluating loc_sum and any changes in the displayed data is highlighted in yellow.

Question? Use the Webex chat facility to ask the Host

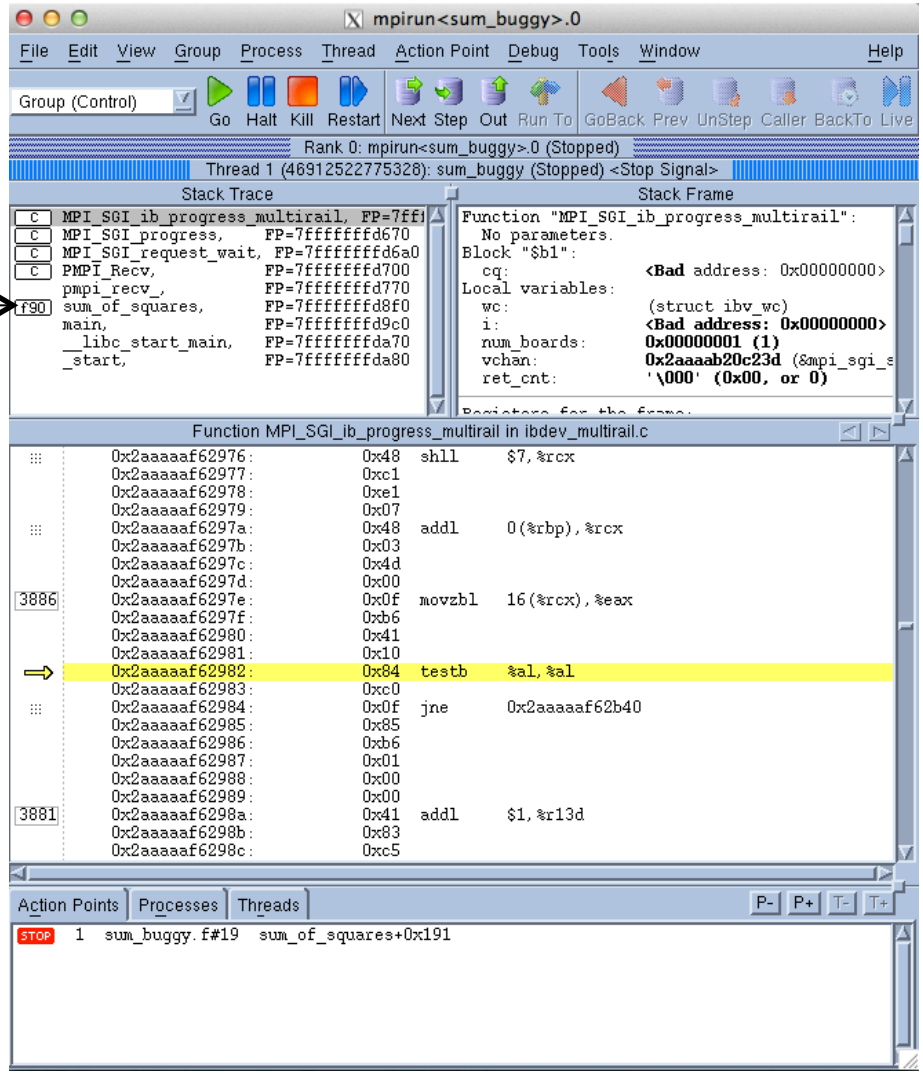
Debugging program hang



Question? Use the Webex chat facility to ask the Host

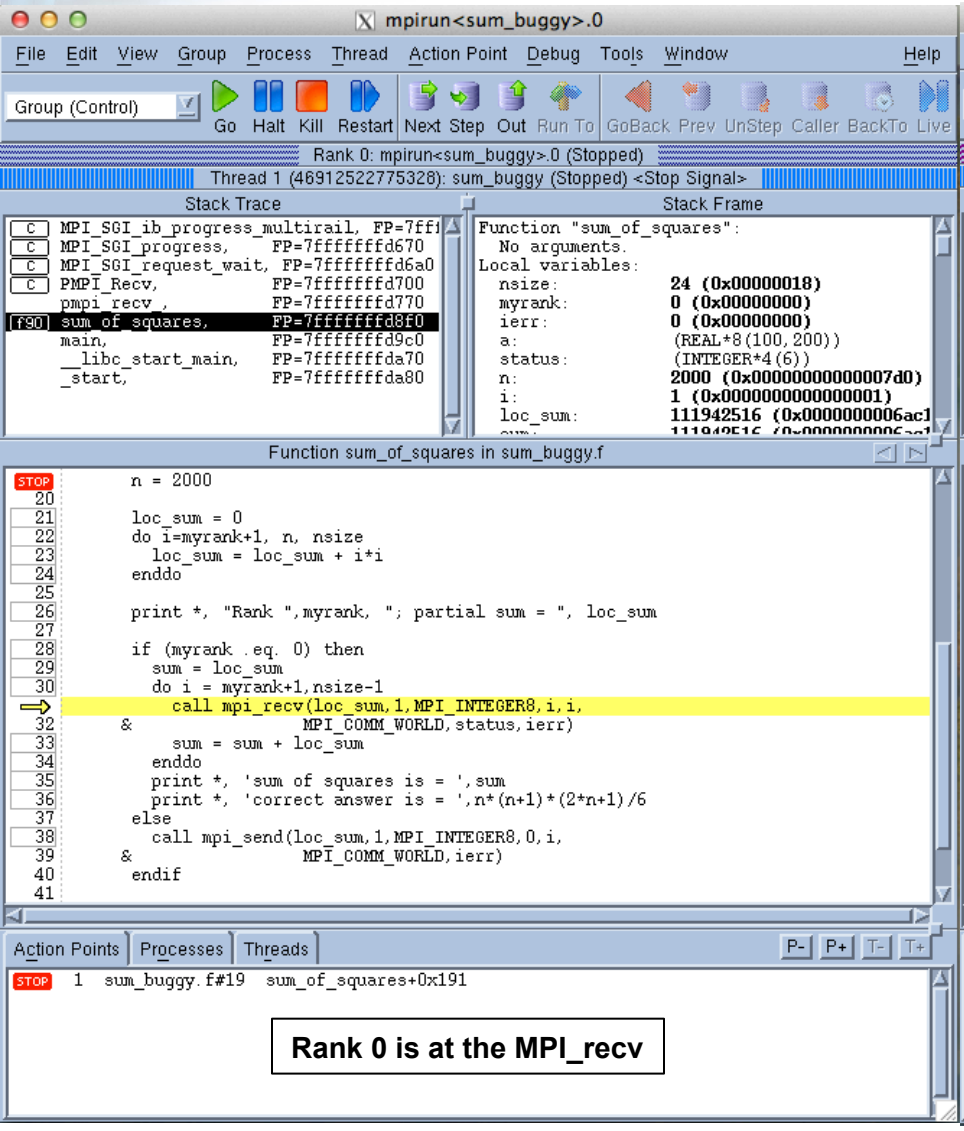
Stack trace

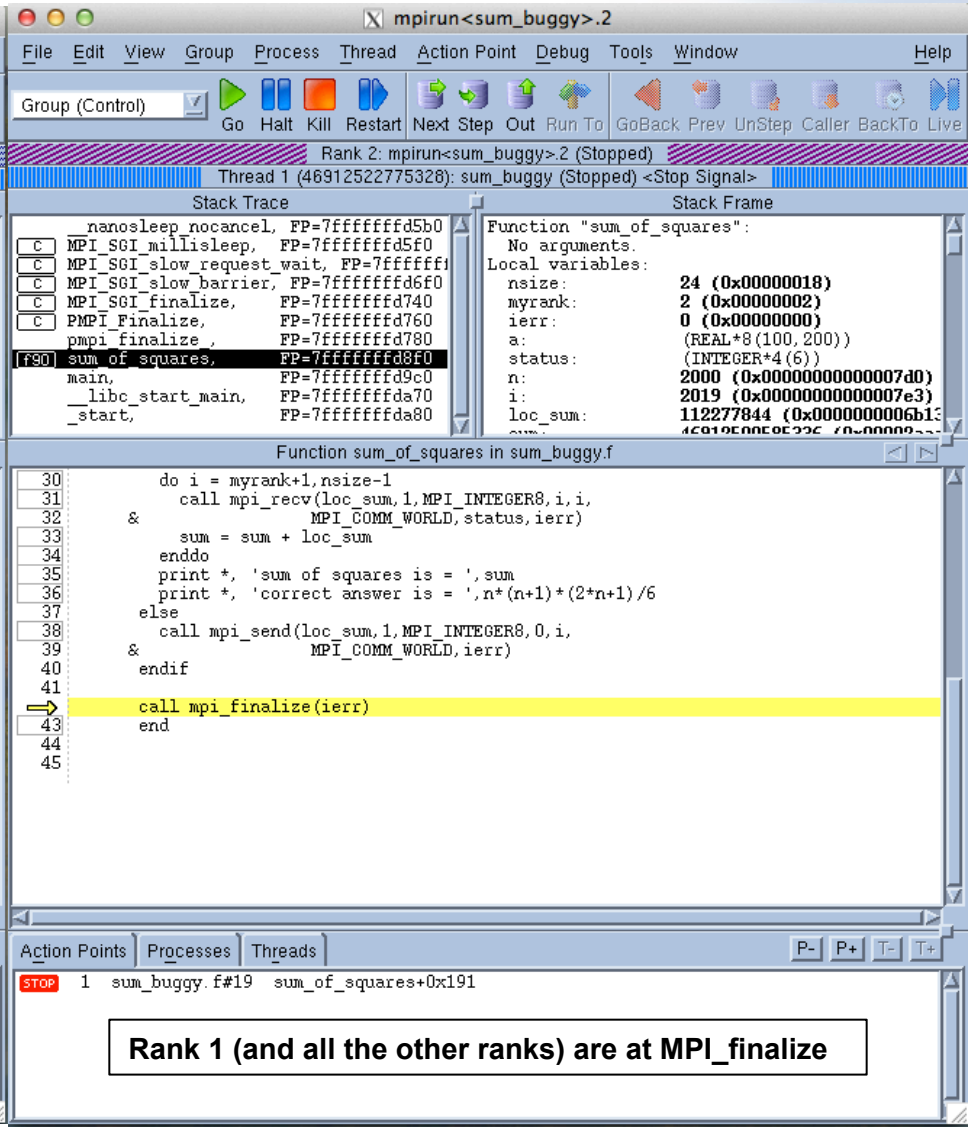
The Source pane shows the binary code deep inside the MPI library. Click on the last stack of your source code to see where it is calling the MPI routine.



Question? Use the Webex chat facility to ask the Host

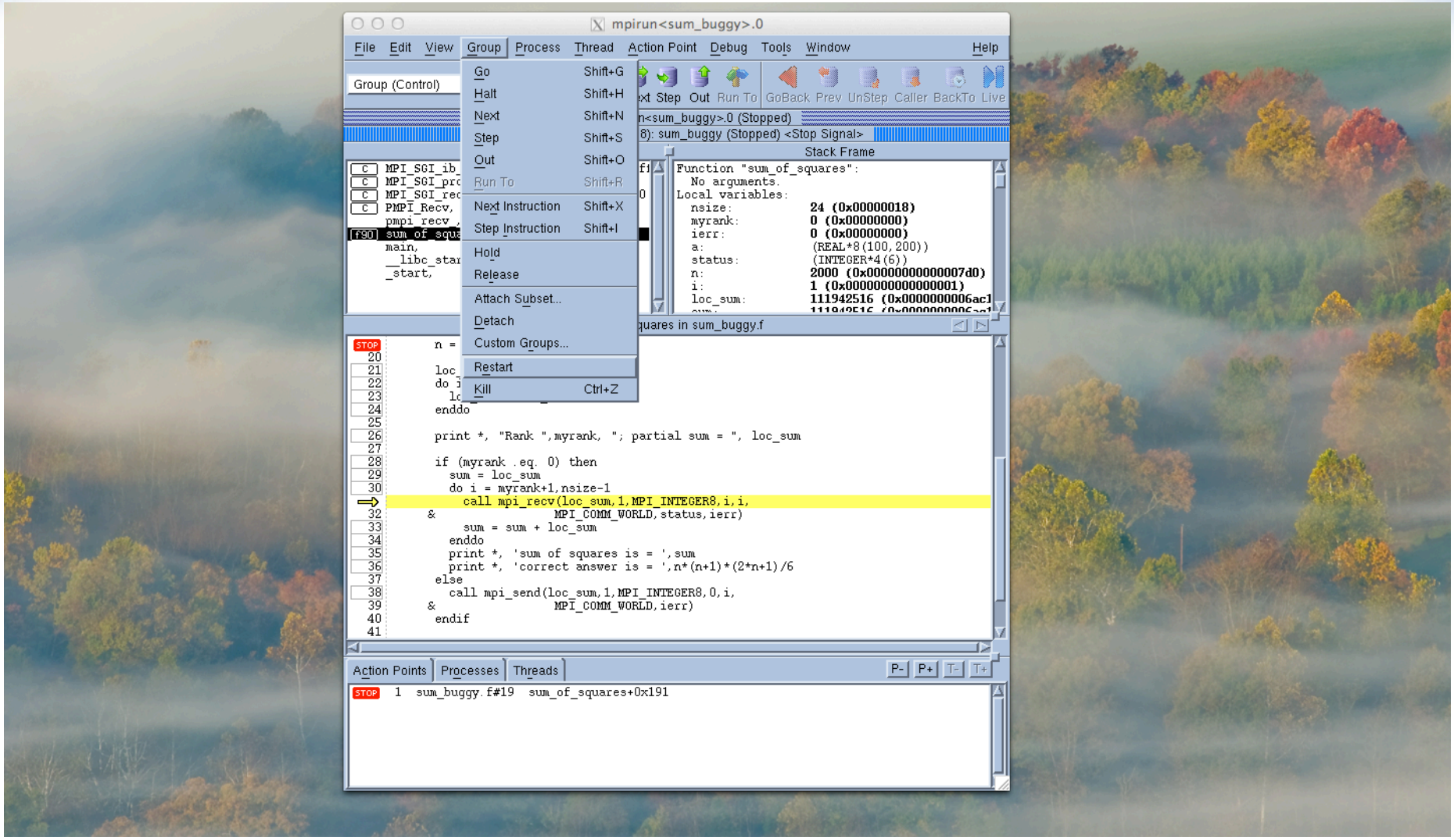
View from two snapshots of the Process Window





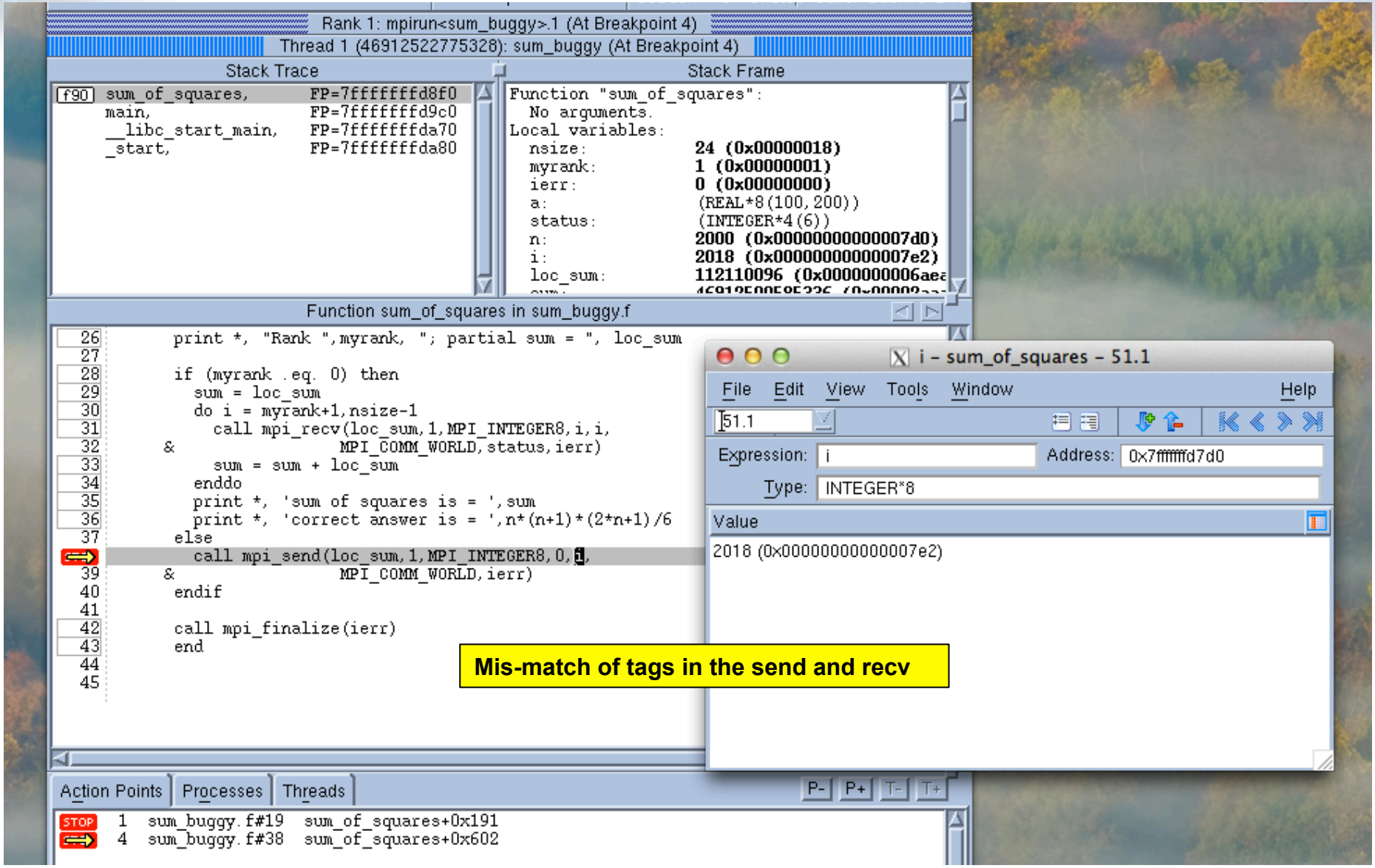
Question? Use the Webex chat facility to ask the Host

Re-run the program and set a breakpoint at the MPI_send



Question? Use the Webex chat facility to ask the Host

Bug Revealed



Question? Use the Webex chat facility to ask the Host

Where to find additional information on using TotalView

Online videos produced by TotalView developers and engineers:

http://www.roguewave.com/products/totalview/resources/videos.aspx

- **Getting Started with TotalView**
- **Debugging MPI**
- **Introducing C++View**
- **Setting Breakpoint References for Threads**
- **Viewing Data Across Threads**
- **Threads Navigation**
- **Asynchronous Thread Control**
- **Debugging OpenMP**
- **C++ STL Type Transformations**
- **Memory Debugging with MemoryScape**
- **Memory Debugging with Red Zones**
- **Deterministic Replay with ReplayEngine**

Online Documentation on Pleiades

```
pfe12> which totalview
/nasa/sles11/totalview/toolworks/totalview.8.9.2-1/bin/totalview
pfe12> cd /nasa/sles11/totalview/toolworks/totalview.8.9.2-1/doc
pfe12> ls
pdf/
pfe12> cd pdf
pfe12> pwd
/nasa/sles11/totalview/toolworks/totalview.8.9.2-1/doc/pdf
pfe12> ls
MemoryScape_Installation_Guide.pdf
MemoryScape_New_Features_Guide.pdf
MemoryScape_User_Guide.pdf
ReplayEngine_Getting_Started_Guide.pdf
ReplayEngine_New_Features_Guide.pdf
TotalView_Installation_Guide.pdf
TotalView_New_Features_Guide.pdf
TotalView_Platforms_and_System_Requirements.pdf
TotalView_Reference_Guide.pdf
TotalView_User_Guide.pdf
```


Question? Use the Webex chat facility to ask the Host